ロバストソフトウェア構成法

武市正人　胡 振江　筧 一彦
情報理工学系研究科 数理情報学専攻

概 要

本プロジェクトは、モデル検査等の形式的手法に基づいてロバストなソフトウェアを構成する方法を確立し、融合プロジェクトを通じてその手法の有効性を実証しようとするものである。

1　はじめに

問題解決のためのプログラムの信頼性を高め、ロバストなソフトウェアを構築するための形式的な方法論を確立するためには、それが基礎としているモデルの検証が必要である。プログラムが仕様に適合するものであるかどうかを検証する技術は 1970 年代から研究されているものの、実用規模のプログラムの正当性を検証することはその複雑さのために不可能であるという現実がある。本プロジェクトでは、ソフトウェアの注目すべき特定の性質を抽出したモデルに対して形式的手法を適用し、モデル検査を通じてロバストなソフトウェアの構築法を確立しようとするものである。

(1) モデル検査システムに関する研究：形式的手法の一つであるモデル検査の手法は、基本的には、有限状態オートマトンにおける状態遷移を検査することによってシステムの検証を行なうもので、各種の効率的な検査手法が開発されている。これらのなかには、「力まかせ」による部分もあり、近年の計算能力の向上によって実用化されたものもある。既存のモデル検査システムを調査するとともに、従来、多く使われてきたハードウェア検査だけではなく、ソフトウェア (プログラム) のロバスト性検査に適合させるための手法を追究する。

(2) モデル検査のためのモデル構築に関する研究：プログラムの検証の困難さは、動的な振舞いの複雑さにある。モデル検査手法によってそのロバスト性を高める本研究の手法は、その振舞いを抽象化し、ある側面から見た性質だけを検査するという抽象解釈の手法により状態数を抑えようというものである。具体的なプログラムに対して、注目する性質が与えられたときに、その抽象化を行なうことはそれほど簡単なものではないが、いくつかの代表的な性質に対する抽象的な領域の設定を通してモデル構築の手法を提示する。

(3) プロジェクト融合によるプログラム検査の実施：他のプロジェクトで対象としているアルゴリズムやプログラムに対して、モデル検査手法を適用し、プログラムの信頼性を高めることを試みる。

上の研究実施にあたっては、この分野の国内外の研究者との交流によって、最初の 2 年間で形式的手法を確立するとともに、実用的な検査システムの応用技術を蓄積し、そのあとで現実の問題を扱うこととする。

2　平成 15 年度の研究成果

本プロジェクトに関連する成果として、昨年度より継続して研究を進めてきたモデル検査技術を用いたプログラム解析法に関する論文を発表した。

2.1　モデル検査によるプログラム解析の自動化に関する研究

昨年度より継続して、「モデル検査技術を利用したプログラム解析の自動生成」を課題として取り組み、論文 [1] を発表した。

この論文では、プログラムの伝統的な最適化手法において利用されるプログラム解析の多くがプログラムのデータフロー上での性質を表現した時相論理によっ

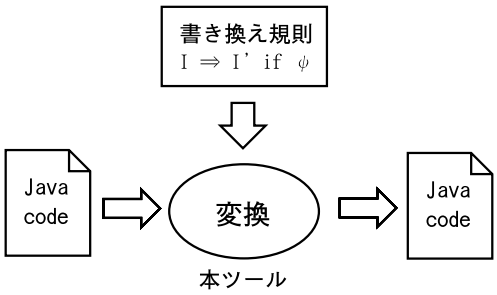

図 1: 処理概要

て記述することができることに着目し、時相論理式で記述されたプログラムの性質の解析処理を自動化するツールの開発とその実用性を論じた。この研究で得られたツールは、現在、一般に使われている実用言語 Java と相互に変換可能な中間言語である Jimple のプログラムが時相論理 CTL-FV による性質記述を満たしているかどうかをモデル検査技術により検証している。そこでは、解析対象の Jimple プログラムと解析したい性質記述の CTL-FV 論理式を入力とし、モデル検査ツール SMV 向けのモデルを生成する。それにより、SMV の実行を通じてそのプログラムが特定の性質を満たすかどうかを解析することができる。本論文では CTL-FV からモデルを構築する方法を中心に述べている。

2.2 モデル検査技法を用いた Java 最適化処理に関する研究

本年度は、前項に述べた研究を発展させ、より具体的に Java プログラムの最適化に対する検討を行なった。そこでは、あらためてプログラム最適化の仕様を宣言的に記述する手法を拡張し、それをもとにして最適化プログラムを実現する方法を追究した。

2.2.1 Java プログラム最適化ツールの設計と実装

本研究は、Java 最適化フレームワーク Soot 上で、利用者によって与えられた最適化処理を施すツールの設計と実装を目的とするものである。

このツールはユーザから与えられた仕様記述に基づき Java のプログラムを書き換えて最適なものに変換する (図 1)。

仕様の記述には命令文の条件付き書換え規則を用い、時相論理で記述された条件を満たすときに命令文を書き換えることによる。このツールを用いることより、不要命令の除去や定数伝播などの一般的なコンパイラ最適化技法の仕様を宣言的な記述で簡潔に表現できることになる。また、その処理を自動で高速に実行することができる。

2.2.2 コード最適化の仕様記述

コンパイラの設計においてコード最適化は重要なパスの一つであり、命令を書き換えて効率のよいコードを生成し、実行速度を向上させたり、コードの量を少なくしたりすることを目的としている。

最適化のためのプログラム解析の従来の仕様記述は宣言的な手法ではなく、自然言語や疑似コードで記述されてきた。しかし、このような記述から最適化処理を自動生成することは難しく、また、与えられた最適化の正当性を証明するのは困難であった。本研究は、プログラム解析の仕様記述を時相論理によって表現して、自動的に最適化を行なう試みようというものである。その枠組で記述された最適化の仕様記述の正当性の検証も比較的容易に行なうことができる。

2.2.3 時相論理 NCTL+FV の提案

Lacey らは 2001 年に、時相論理 CTL に過去の時制を扱う演算子を加え、さらに自由変数を導入して拡張した時相論理 CTL-FV を提唱し、伝統的なプログラム最適化の仕様の多くが CTL-FV による条件記述とその結果を用いた命令文の書換えによって記述できることを主張した。時相論理 CTL は分岐する時間のモデルの状態を表わす論理式で記述するが、従来の論理式に加えて時間に関する性質を記述する演算子が追加されている。また、CTL-FV は自由変数を扱うことができるので、最適化の仕様に十分な記述力を持つといえる。しかし、その記述力の増強は、現存する一般的なモデル検査のツールで扱える範囲を越えることとなり、CTL-FV による記述に対するモデル検査の自動化には難しい面がある。

昨年度に、われわれは、時相論理を用いたプログラム解析を行なう解析器生成ツールを実現した。そこで

は、解析対象言語を Jimple とし、Soot の Jimple パーザを用いて実装している。 Soot は Java コードの最適化フレームワークであり、Java コードの解析や変換などの目的に沿ったいくつかの中間表現を生成するとともに、その上での基本的な操作を行なう環境を提供している。Jimple は型が付与された 3-address 表現であり、最適化処理に容易な形式をしている。昨年度の実装ではモデルを構築して、既存の SMV モデル検査ツールを用いてモデル検査を行なうという一連のプログラム解析処理を自動的に行なうものであった。しかし、この実装では過去時制などの拡張のない CTL に自由変数拡張を施した時相論理を性質の記述に用いている。また、自由変数は Java の局所変数のみ取ることができ、述語も限定されたものであった。

これに対して、本研究では、CTL-FV のようなプログラム解析仕様を表すための時相論理として NCTL+FV を提案し、それに基づく最適化のための解析ツールを実現した。NCTL+FV は 2000 年に提案された時相論理 NCTL に自由変数の概念を拡張したものである。NCTL は CTL に対して過去の時制を扱えるように拡張されており、性質の記述をより自然に行なうことができる。NCTL+FV は CTL-FV と比較すると, 過去時制の扱いに若干の制約があるものの、プログラム解析に必要な表現能力はほとんど損なわれていない。さらに、NCTL+FV で与えられた宣言的な最適化の仕様から自動的に最適化の処理を行なう環境を得ることができるという特長がある。

2.2.4 モデルの生成

仕様記述に対する対象プログラムに関しては、そのモデルを生成して、モデル検査によって仕様との整合性を検査することになる。その一例を図 2 に示した。これは、メソッドのコントロールグラフからモデルを生成する概要を示すものである。

2.2.5 最適化ツールの実装と実験

本研究で実現したツールは Java 最適化フレームワーク Soot 上で実装されており、NCTL+FV から CTL+FV への変換器を経由して、組込みの CTL モデル検査器によるモデル検査を行ない、その結果を用いて Jimple 上の命令文の書換えを行なっている。

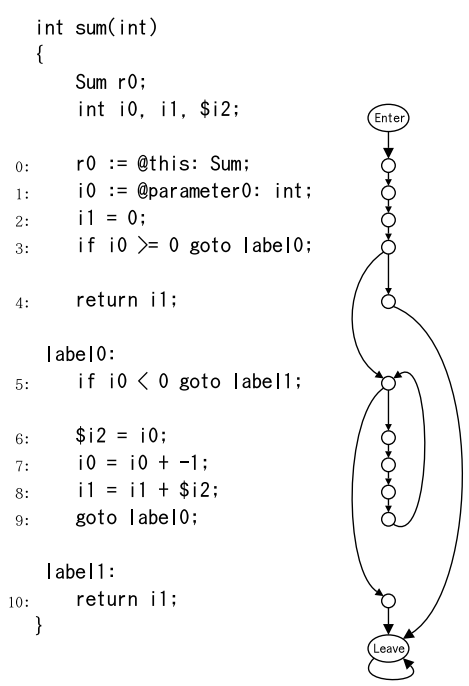

図 2: コントロールフローグラフからモデルを構成

本ツールを用いて、実際に以下の Java コードに対して、不要命令除去と定数伝播の最適化を行なう際の時間の計測を行なって、実用性を評価した。

- **Nothing-app**
  空のクラス Nothing と、その実行時に要求される GNU Classpath ランタイムライブラリ群全体。これは標準の実行環境に用いられるライブラリのなかに、最適化の余地があるかどうかの評価につながる。

- **BigInteger**
  JDK1.4.1java.math.BigInteger クラス。クラスファイルサイズが比較的大きいものの例。

実験環境は以下の通りである。

- コンパイラ: GCJ 3.3
- 実行環境: SableVM 1.0.9, GNU Classpath
- CPU: Pentium M 1.3GHz
- メモリ: 512Mbytes

実験結果が表 1 に示された。数十から数百 KB 程度のクラスファイルの解析でも、最適化処理が数秒から

表 1: 実行結果

対象	Nothing-app	BigInteger
クラスファイル数	30	1
ファイルサイズ [bytes]	151,282	30,467
メソッド数	801	108
Soot の消費時間 [秒]	96	44
不用命令除去		
除去した命令数	21	9
要した時間 [秒]*	4(100)	6(50)
定数伝播		
変換した命令数	0	0
要した時間 [秒]*	1(97)	17(61)

* 括弧の内側の数値は処理全体の時間, 括弧の外側の数値は Soot 自体の処理時間を差し引いた時間である.

十数秒で終了していることがわかる。この速度は実用的には問題なく、本ツールの有効性を示している。

どちらの対象コードも不用命令を検出しその除去を行なったが、広く流布しているライブラリのコードの中で、最適化し尽くされていない部分を検出し、その部分に最適化を施すことができたといえる。その一方で、定数伝播の最適化については、書換え対象となるコードは検出されなかったので、すでに既存の手法で最適化されているものと考えられる。

本研究では、NCTL+FV によって、いくつかのコンパイラの最適化の記述を行なうことができることを示したが、さらに多くの例を調査して、このツールが扱うことができる範囲を明確にする必要がある。また、ユーザが新たな述語を必要とした時に、それを容易に追加することができるような仕組みを実装して、より汎用的なシステムを構築することが今後の課題である。

また、本実装では CTL 組込みのモデル検査器を利用してモデル検査を行なっているが、この検査器の効率はかならずしもよいものではない。この点を改善するために、組込みのモデル検査器ではなく、SMV などの既存の効率のよいモデル検査器を利用することも考えられる。しかしその際には、本実装で NCTL から CTL への変換時に行なっている部分木の共有による式の圧縮が無効になるため、効率の低下を招く恐れもある。したがって、現在の組込みのモデル検査器を、BDD などを用いて高速化処理を行なうように改良することも考えられる。

現在の実装は、過去時制を扱うために、用いる時相論理を NCTL+FV とし、NCTL+FV から CTL+FV への変換を経由しモデル検査を行なっているが、この変換処理の高速化や生成される CTL 論理式の長さの効率化は考慮していない。これらの処理の効率化も今後の重要な課題であるといえる。

3 おわりに

本プロジェクトで扱っているモデル検査手法によるロバストソフトウェアの構成法については、近年、研究が盛んになってきているものの、実用的なソフトウェアに対してその手法を適用する技術はまだ緒についたばかりである。昨年度と本年度は、プログラム最適化に関する手法の開発とツールの実現を中心に研究を進めてきた。今後とも、この課題を追究する計画である。

4 発表論文

1. 山岡 裕司, 胡 振江, 武市 正人, 小川 瑞史. モデル検査技術を利用したプログラム解析器の生成ツール. 情報処理学会論文誌 Vol.44 SIG13(PRO18), pp.25-37, 2003

2. 番 伸宏, 胡 振江, 筧 一彦, 武市 正人. Java プログラム最適化の宣言的記述とその効率的な実装. 日本ソフトウェア科学会 PPL2004. (発表予定).